HAAGA-HELIA
University of Applied Sciences

Error rates reduction in handwritten digits classification using the MNIST data with Artificial Neural Networks

Daniel Pereira de Freitas
Alexander Shkunov

Bachelor's Thesis
Degree Programme in Business Information technology
November 2014

HAAGA-HELIA
University of Applied Sciences

Abstract

November 2014

Author(s)
Daniel Pereira de Freitas
Alexander Shkunov

Degree programme
Business Information Technology

Report/thesis title
Error rates reduction in handwritten digits classification using the MNIST data with Artificial Neural Networks

Number of pages and appendix pages
36+4

Supervisor
Kari Silpiö

The following paper is an introductory level research on Artificial Neural Networks and Machine Learning. The main objective of the thesis is to reduce the error rates in handwritten digits predictions using Artificial Neural Networks. The dataset used was the famous MNIST dataset, which is publicly available for use.

Throughout the research, several ecosystems have been tested and their results catalogued in the Results sections towards the end of the thesis. The main tools used were R, Octave and MATLAB. It has been found that out of the 3 aforementioned tools, R is the one with the worst performance whereas Octave and MATLAB both show a significant performance improvement. Finally, the thesis was finalized with the MATLAB ecosystem due to a mix or performance plus convenience, which allowed for very quick and efficient prototyping.

As for the data predictions, the dataset was divided into a proportion of 60/20/20 for training, validation and test sets respectively. The results were catalogued throughout 5 implementations each building on the top of the previous one until an accuracy plateau was reached. Said plateau was found at the 97.6% mark thus allowing to conclude that error rates can indeed be decreased through Artificial Neural Networks.

Additionally, it has been found that there are several ways one could theoretically use in order to increase the ANN's performance. For one, the authors were able to systematically decide the ideal amount of hidden units needed as well as the regularization.

The paper also discusses the vital role, data processing plays in the overall outcome, performance and efficiency of the algorithm. As a matter of fact, the algorithm is only one part of a much bigger picture which is Machine Learning.

Keywords
machine learning, artificial neural networks, software development, data mining, mnist

Table of contents

Table of figures

1 Introduction

1.1 General Introduction

Machine Learning is a multidisciplinary field, which positions itself in the intersection of statistics, computer science and engineering. It is a relatively “new” area, which has been gaining significant traction over the last few years.

According to Mitchell (1997, 2), Machine Learning can be formally defined as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". In other words, Machine Learning gives a piece of software the ability to learn, to perform a given task without being explicitly programmed and improve its performance over each iteration.

In the real world, the applications for machine learning are vast and numerous. Nowadays, it is common to find it in big companies such as Google, Netflix or Amazon. Amazon, for example, makes intensive use of machine learning to profile their users and based on that suggest items that said customer would possibly be interested. Google also uses machine learning to show relevant search results based on the profile of a given user.

Briefly, Machine Learning is currently a very prolific field and possibly one of the next big trends in the IT landscape. Even if not explicitly noticed, thousands of people already interact with Machine Learning in some way. On top of that, it also facilitates and furthers several fields, which may not be explicitly linked to it, but still use it to an extent.

This thesis in particular is concerned with reducing the error rates in the recognition of handwritten digits using Artificial Neural Network. This paper will deal with the technical aspects of Artificial Neural Networks.

1.2 Research Problem, Questions and Objectives

In this paper, the focus is to improve the performance of a given algorithm over a dataset. Evidently, said task is not as trivial as it may initially seem. For one, the amount of data collected plays an important role on the overall outcome of the predictions, in other words, the more data one has access to the better the results will be.

The main objective of this research is to reduce the error rates in hand-written digits recognition by applying Artificial Neural Networks using different approaches and methods.

Additionally, it is the aim of this study to answer the following questions:
Can Artificial Neural Networks classification error rates be decreased?

- How can it be achieved?
- What are the most efficient methods?
- What is the impact of these methods on the final accuracy?

The final deliverables in this paper will include:

- Source Code (Published in Github)
- Documentation
- Screenshots taken throughout the project

1.3 Research domain and scope

This paper will be centered on one particular known algorithm and a predetermined single dataset. Anything concerned with attaining the results such as programming language, Integrated Development Environment, algorithm and so forth will be considered part of the domain of the thesis.

It is important to emphasize that although the aforementioned items are part of the domain of the thesis that does not mean that they will be deeply discussed in this paper. However, resources may be provided in case the reader wants to familiarize him/herself with the topic.

2 Theoretical background

2.1 Machine learning

Even though the concept of “learning” algorithms is rather old, implementation of said algorithms has only been possible in the last decade thanks to the exponential growth in technology which resulted in multicore CPUs and vastly elevated processing power. As mentioned in chapter 1, not only machine learning is used and developed in the academic environment but also in more practical settings. The area itself has also been garnering a lot of attention from programmers, data analysts, physicists among others due to its many applications in their respective fields. As a result, there are currently more books, theses, articles and open source code than ever before. (Hamilton 2014.)

Several influential books have been written about machine learning. One of the most influential books on the field is titled: “Machine Learning”, Tom Mitchell wrote it in 1997. Moreover, several other influential papers have been written on the topic, some of them date back to, as far as, 1956 such as “An Inductive Inference Machine” by Ray Solomonoff. Said papers and books, are mostly concerned with implementing efficient algorithms to solving learning problems.

According to the scikit-learn[1] page, a given learning problem takes into consideration a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number it is said to have several attributes or features.

To simplify matters even further, the term “learning” can be boiled down to three basic steps:

1. Representation
2. Evaluation
3. Optimization

Every single machine learning algorithm can be distilled into this basic “pipeline”. (Domingos 2012, 1.)

Evidently, there are many approaches one could theoretically take to solve a learning problem. That being said, only one of said approaches will be discussed on this paper: Supervised Learning.

[1] http://scikit-learn.org/stable/tutorial/basic/tutorial.html

2.1.1 Supervised learning

Supervised learning is, by definition, a machine learning task that has as its' primary objective to infer a mathematical function based on a labeled dataset known as the training data. The dataset analyzed by the algorithm consists of two pieces of data where the features (input) are usually represented as X and the labels, also known as, supervisory signal, which is the desired output as Y. After formulating a generalization based on the dataset, the algorithm determines the hypothesis value H for unknown data.

Supervised learning can be further subdivided into more than one type: Regression (Predict continuous value output) and Classification (Discrete Value Prediction). In the first one, the label value has no known limitation regarding its range (e.g. any real number). In the latter one, on the other hand, we have only a limited range of possible values (e.g. A, B, C).

2.1.2 Classification (Discrete value prediction)

As explained in subchapter 2.1.1, Classification works with a predetermined range of possible values for a label. The classification method makes use of a function approximation based on a hypothesis, which can be further represented as a decision boundary line. Decision boundary is an abstract line (surface) that partitions the classes in a given space (one or more dimensions).

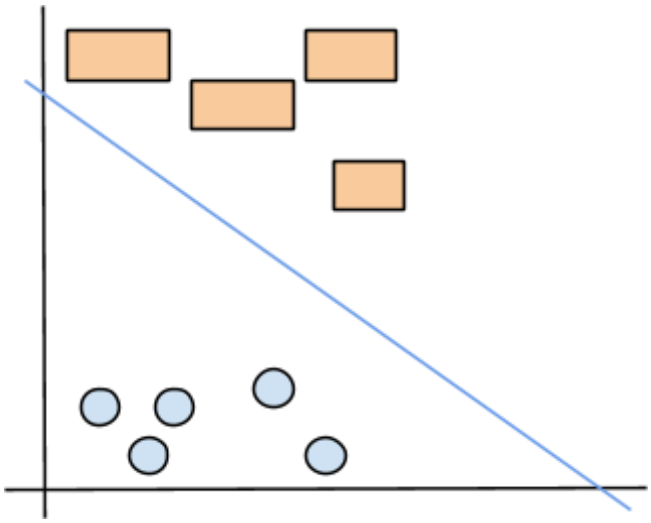

Figure 1 Decision Boundary Example

In figure 1, depicted above, the blue line represents the decision boundary, which separates circles from rectangles. Incontestably, this is a line represented in a Cartesian plan,

but said partition can happen on more than two dimensions in which case we would need a different representation for the boundary.

2.1.3 Optical character recognition

OCR (Optical Character Recognition) is, as the name suggests, a technology to recognize an image file or bitmap file whether it has been scanned, handwritten, typed or printed. Through those means, one can, with OCR, obtain a text file, which can be edited by the computer.

2.1.4 Artificial neural networks

According to Mitchell (1997, 81), Artificial Neural Networks provide a general, practical method for learning real-valued, discrete-valued and vector-valued functions from examples. Referring to the same author ANN was successfully applied in speech recognition, robotics and computer vision.

Artificial neural networks were inspired by a biological concept of how our own brain works, where there are several neuron units, which one by one process some input and provide some output. In machine learning, when we talk about classification using neural networks, each unit by itself is a logistic regression unit. Respectively logistic (sigmoid) function is used to provide hypothesis H.

An interesting parallel one may trace between Biological Neural Networks and Artificial Neural Networks is that both theoretically learn from example, like a kid who would learn to identify a dog by seeing examples of dogs; an Artificial Neural Network would also learn to identify a dog by seeing examples of dogs.

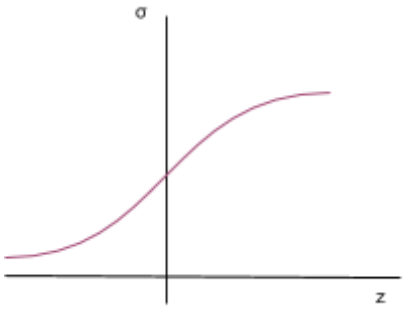

Figure 2 Sigmoid function example

The sigmoid function in figure 2 can be represented as follows

$$\sigma = \frac{1}{1 + e^{-z}}$$

where Z is the multiplication of Network weights by object features.

In the basic model (network architecture), the amount of input units corresponds to the amount of features and the amount of output units corresponds to the amount of classes (possible discrete-value labels). The hidden layer has as its purpose to transform the input into something that can be used by the output layer. Assuming a network has three inputs, two hidden ones plus an output one - It can be depicted as follows.

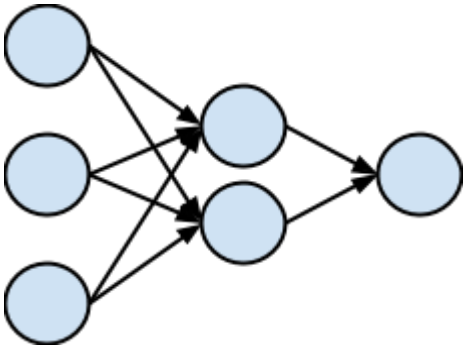

Figure 3 Simple example of ANN with one hidden-layer

As it can be seen in figure 3, each layer unit connects with each unit of the next layer, and the last output unit provides the final hypothesis based on the input from previous layer units. With this representation, each connection between two units then is a single weight W, or Theta in notation. So for example, here we then have 6 weight values between input and hidden layers, and then two weight values between hidden and output layer.

Particularly, consider this example (created by the authors for the sake of simplicity): let us say that for some reason we want to predict whether a student will pass a course in university. We can define two basic criteria for a specific course; for example, more than 50 % in the exam and at least 80 % of participation is needed. Considering these criteria, we can define what our target function is as figure 4 shows.

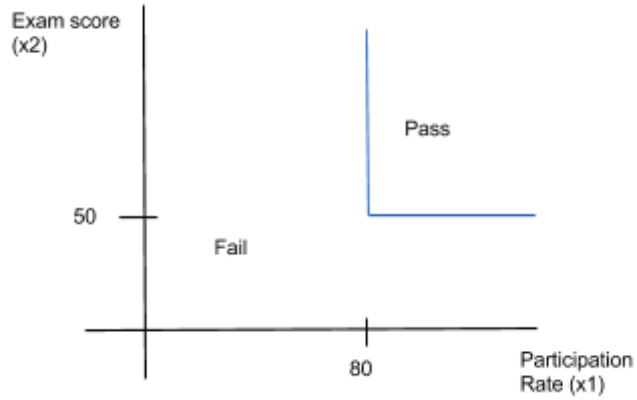

Figure 4 Target function represented by decision boundary (blue line) for student pass criteria

With this particular example we can say, that target function is

$$g = \begin{cases} 1\ (true) & \text{if } x_1 \geqslant 80\ and\ x_2 \geqslant 50 \\ 0\ (false) & \text{if } x_1 < 80\ or\ x_2 < 50 \end{cases}$$

However, in some cases we cannot know what the particular criteria are and thus cannot define a target function. So let us say some professor does not specify the conditions needed to pass the course. In that case, we start to collect data to know which students passed the course and what were their attendance and exam score. Eventually, after collecting the data, we run the neural network algorithm, which generalizes from the given data and approximates the function. As it can be seen in figure 5 below, the red circles represent students who did not pass, green - those who passed and blue line, in this example, is the decision boundary, which represents an approximation of the function.

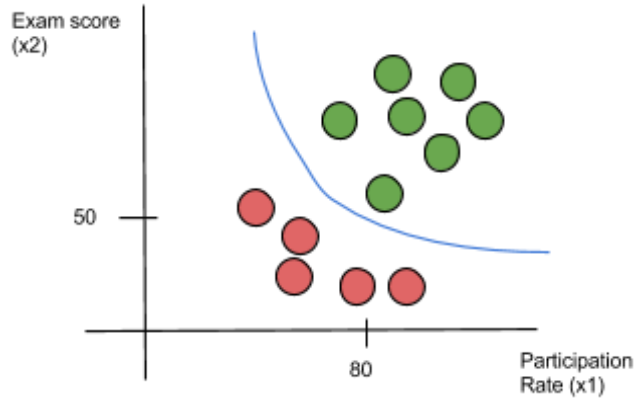

Figure 5 Function approximation example

In any case, referring back to the previously introduced sigmoid activation function, the hypothesis H is defined as follows

$$H_\Theta(x) = \frac{1}{1 + e^{-(\Theta^T * X)}}$$

The approximation can be explained as follows, whenever multiplication of weights and features is more than or equal to zero, then the hypothesis is positive, e.g.

$$H_\Theta(x) = \begin{cases} 1\ (passed\ the\ course) & \text{if } \theta_1 x_1 + \theta_2 x_2 \geqslant 0 \\ 0\ (failed) & \text{if } \theta_1 x_1 + \theta_2 x_2 < 0 \end{cases}$$

With that being said, we come to some interesting thoughts, such as how good our approximation should be to be useful, how can we estimate the error of the approximation without being able to define the target function and more importantly how do we improve the accuracy and ensure that it closely matches the unknown target function. These questions, in a more or less specific way are the target of our research and will be more descriptively overviewed in following chapters.

2.2 Problems that affect accuracy

2.2.1 Bias/Variance

Quoting Fortmann-Roe (2012): “The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict. “ - Whereas the error due to variance - “is taken as the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the model.”

Figure 6 depicted below illustrates the concept. The center of the target is a model that predicts the correct value. As the predictions move away from the bull’s eye, they get progressively worse.

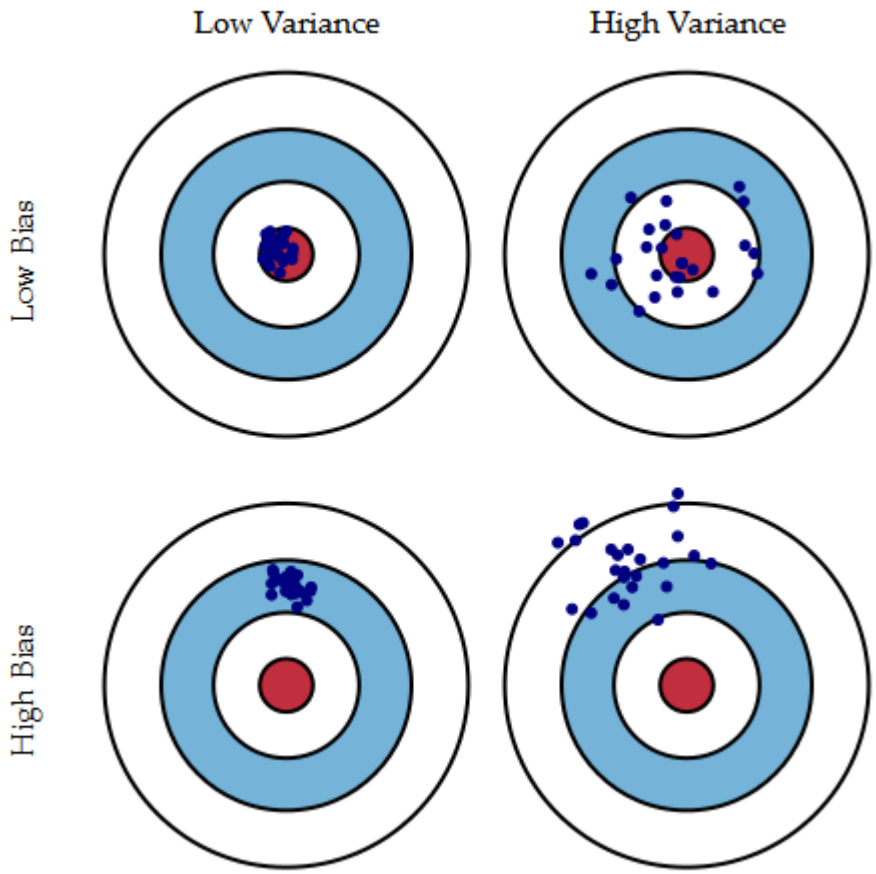

Figure 6 . Conceptual representation of bias/variance states

2.2.2 Failed optimization (Cost function)

Referring back to chapter 2.1, any learning consists of the following: Representation, evaluation and optimization. So the process can be described in the following steps. First, we randomly plot a decision boundary and based on that we calculate the cost function, a measure that tells us how far away our hypothesis is from the optimal hypothesis. Then using the optimization algorithm, we find the decision boundary that has the lowest possible cost function value. On some instances, when the cost function is not convex or the chosen optimization algorithm was not accurate - the program will fail to find an optimal value - global minimum. However, that may implicitly lead to a high bias.

2.2.3 Curse of Dimensionality

The term Curse of Dimensionality was originally coined by Bellman in 1961 (Domingos 2012, 4). It refers to the fact that, on the contrary, of what one would instinctively believe:

adding more features to a classifier thus increasing the number of dimensions will not improve the accuracy of the learner. Actually, it opens room for more errors. Figure 7 depicted below illustrates this in a graph.

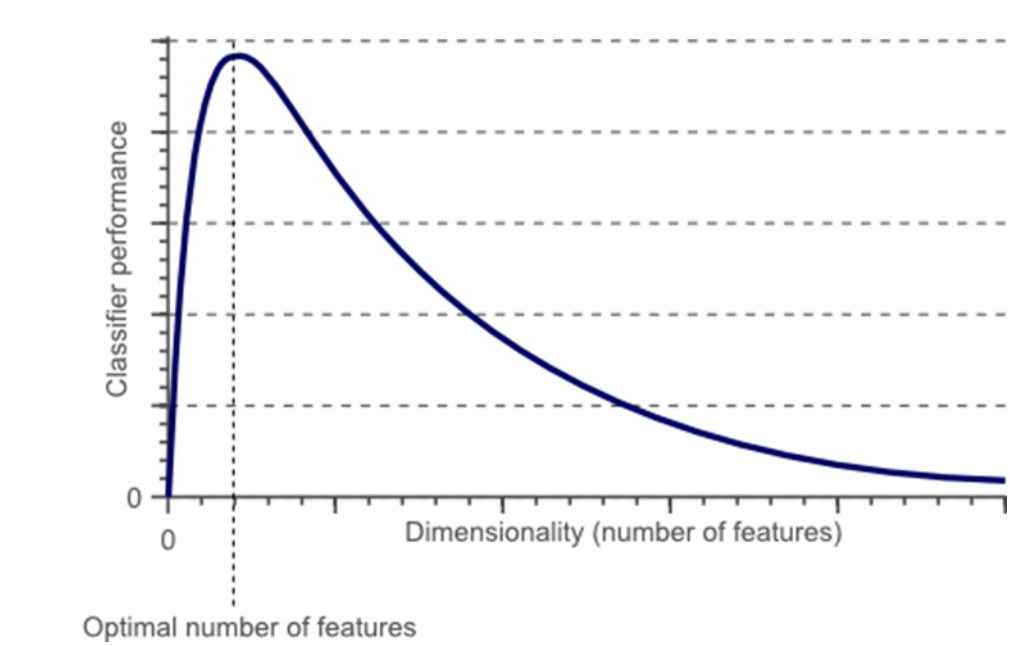

Figure 7 . Performance x Dimensionality graph.

The curve shows that adding more features may help the performance at lower dimensions (when the number of features is limited), but it steadily decreases as more features are added. (Spruyt 2014.)

The paragraph above, may lead to very natural doubts and confusion for, in this case, common sense and intuition may lead to incorrect results. In order to minimize the problem described below, one has to, very successfully, be able to extract the most relevant features for a given data set. (Domingos 2012, 5.)

2.3 Learning algorithm debugging and model optimization

In order to better illustrate this section, let us start with a very simple example. Suppose, a developer wants to build an anti-spam feature for e-mails. The developer then proceeds to carefully collect the necessary data that will be used to label a given message as a spam. Finally, the developer finds out that his algorithm is getting a 20% test error, which is unacceptably high. This section will deal with possible solutions that can be applied to debug and optimize the learning algorithm.

Considering some of the problems mentioned in subchapter 2.2, one could try to improve the algorithm. For example, one could potentially try to get more training examples, increase or decrease the set of features or even change some of the features altogether.

Even though the approach described above may be successful, it still is time consuming and it depends a lot on luck.

Evidently, one may prefer a more systematic approach to tackling such issues. Several ways can be used to diagnose and identify the source of a problem in a learning algorithm, which can be used for model optimization.

2.3.1 Bias vs. Variance Trade-off

In general, it should be very simple to identify either high bias or high variance on a learning algorithm. When one talks about high bias both the training error and the test error will be quite elevated. On the other flip of the coin, one may have a good training fit, but a high test error due to the algorithm's failure to generalize from the training set.

Figure 8 below illustrates the concept more clearly.

Figure 8 Learning curve example (Ng)

The picture on the top (error x training set size) shows high variance. One can clearly see that there is a significant gap between the training error curve and the test error curve. In order to fix high variance one may try to get more training examples or a smaller set of features, which indirectly results in increasing the bias.

The second picture depicts a high bias, the gap between the training curve and the test curve is quite narrow. On top of that, the error is quite high in both cases. In order to fix that one should either try a larger set of features or pick a different set (of features).

An optimal function will always contain both, variance and bias, to some extent when optimizing a learning algorithm one has to find always the balance between variance and bias. Decreasing the variance will indirectly raise the bias and vice-versa. (Ng.)

2.3.2 Cost Function Evaluation

Learning algorithms, as explained previously, are improved during the course of several iterations. However, it can happen that the cost function fails to be correctly optimized. It is up to the developer to detect this issue. One can do so by plotting the corresponding curve based on the previously attained results and iterations.

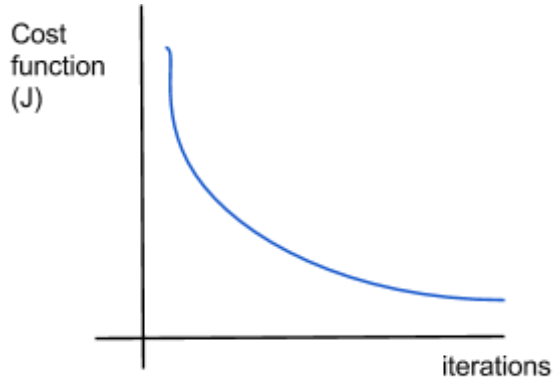

Figure 9 Cost function over iteration example

The curve depicted in figure 9 shows the ideal curve. The cost function should always decrease as the number of iterations increase. Incontrovertibly, the curve, at some point, the slope will decrease but it is unlikely, although within the realm of possibility, that the curve will touch the x-axis.

If a discrepancy is identified, there are two possible scenarios: either there is a bug in the implementation of the algorithm itself or the algorithm chosen by the developer does not suit the task at all, in which case he/she should consider implementing a different algorithm.

2.3.3 Model calibration and optimization

If one wants to follow systematic approach for problem solving, it always makes sense to start with the easiest and simplest model, making it more complex on demand.

Particularly, let us say a developer has a dataset with limited features and he is facing error due to high bias, one of the potential ways to solve this issue might be increasing polynomial of the function by raising features into a power, which will eventually lead to a non-linear predictor as figure 10 illustrates.

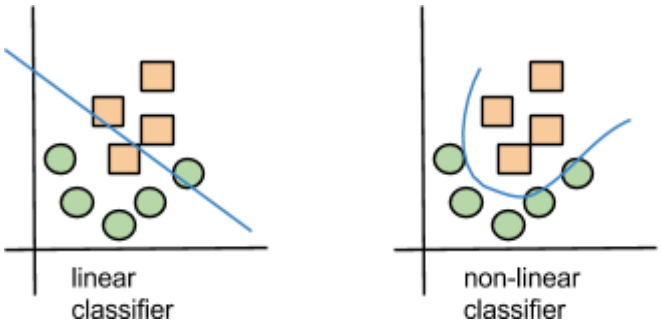

Figure 10 Decision boundary (blue line) for linear and non-linear classifier

For further clarification, figure 10 indicates two distinct classifiers: Linear and Non-linear. A linear classifier is, by definition, in a 2-dimensional space, a straight line. In a multidimensional space, a hyperplane. Any non-linear shape, on the other hand, represents a non-linear classifier, as the name explains (e.g. curves, ellipsis). (Manning, Raghavan & Schütze 2008.)

However, as mentioned earlier, there is a concept of trade-off; this specific solution will only be helpful up to a certain point. For that purpose, it can be quite useful to build several models, plot their training fit and test their accuracy as shown in figure 11.

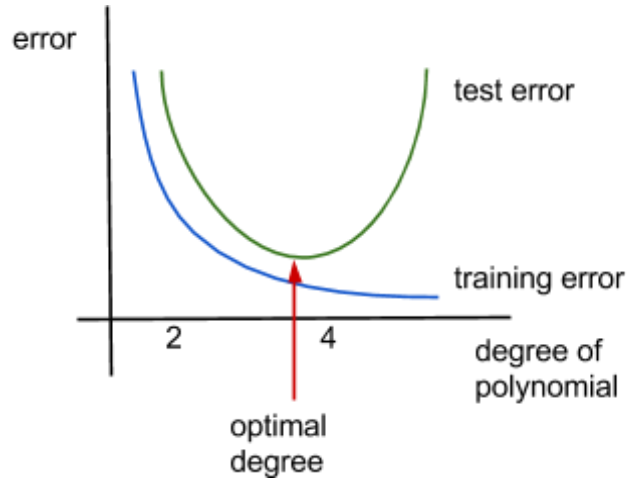

Figure 11 Curves of training error and test error with respect to a degree of polynomial

Consequently, speaking about concrete algorithm of neural networks the similar effect of increasing the model complexity in order to reduce error due to high bias - different amount of hidden units might be carried out as shown in figure 12.

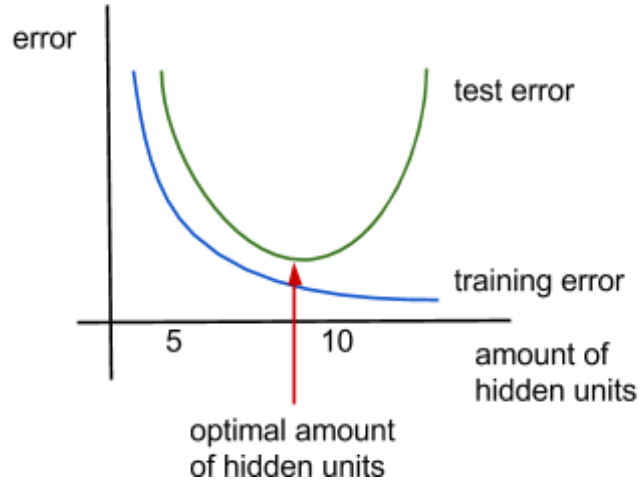

Figure 12 Curves of training error and test error with respect to amount of hidden units in artificial neural network.

According to Jeff Heaton, artificial neural network with one hidden layer can approximate any function that have continuous mapping from one finite space to another. Overall, there is no theoretical reason to use more than two hidden layers. (Heaton 2008.)

For additional calibration, it is usually sensible to use regularization, in other words, the introduction of additional information to avoid over-fitting by penalizing the complexity. Even though on a high level of idea increasing regularization value will reduce model complexity, which can also be achieved by reducing the amount of hidden units - in reality, penalizing the cost function provides a slightly different effect. Instead, regularization works in a

more accurate manner with a smaller scale of change, which allows drilling down an acceptably accurate model. Figure 13 illustrates the process of fitting several neural networks to find a regularization value that gives minimal error rate. (Srihari.)

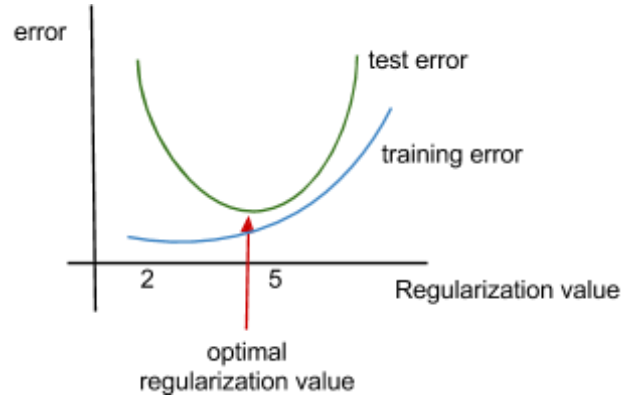

Figure 13 Curves of training error and test error with respect to regularization value.

Concisely, having proper evaluation of metrics and checking training fit and test error at different states of the model complexity can help to calibrate it by bringing it to an optimal state in a time-efficient and systematic fashion. Figure 14 illustrates an example of model calibration and its effect to decision boundary.

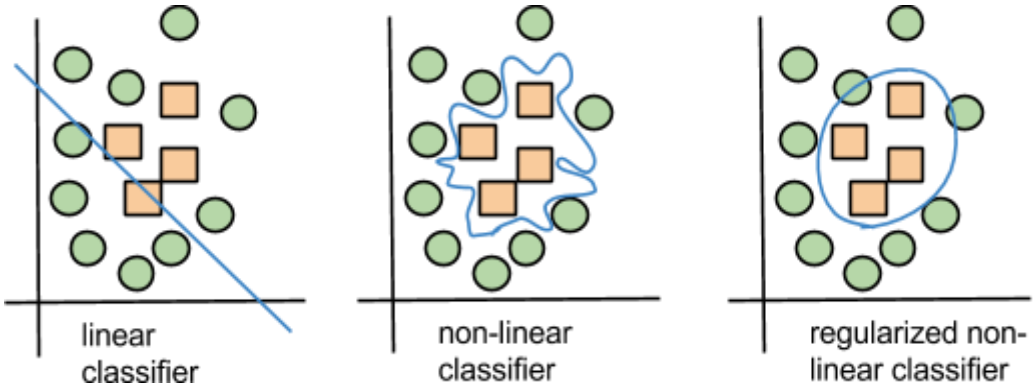

Figure 14 Model calibration - effect of changes.

Indisputably, several different diagnostics could possibly be performed to optimize any given model. The ones presented here are the ones deemed the most appropriate ones within the context of this paper.

This section, finalizes the first part of this paper, the theoretical background. The information presented up to this point will be used as a basis for the upcoming parts of the thesis.

3 Empirical part

3.1 Target of research

To define a target of the research we might align it against machine learning definition, where task T would be recognizing and classifying handwritten digits within pixel grey-scale density representation of images, experience E would be collection of images or their numeric representation, and performance P would be accuracy with which program is able to correctly classify given image. Based on that, the target is to improve P as much as possible using various methods that were described in the theoretical background of the paper and to see which of them would work the best.

3.2 Objectives, problems, development tasks

The objective is to reduce error rates of Artificial Neural Networks (ANN) on a given dataset. One problem posed by artificial neural networks is the inherent difficulty regarding the interpretation of its models. Development tasks include implementing basic algorithm, various metrics calculations for evaluation of algorithm performance, automatic model calibration and optimization by trying different parameters and finding optimal ones as well as possible data permutations in case they will affect accuracy.

3.3 Methodological choices

Artificial Neural Network was selected among other algorithms due to its high performance in the field of computer vision which optical character recognition (OCR) is part of. The attached bibliography offers further explanation on the matter.

Additionally, Artificial Neural Networks can address some of the issues present in different algorithms. For example, Artificial Neural Networks should have a higher accuracy when evaluating a noisy sample.

To comply with the objectives explained in the theoretical background there will be the need for an establishment of an algorithmic evaluation. For this reason, it is essential to proceed in a systematic and scientific fashion.

3.4 Research dataset

For this research, the authors have decided to use a ready dataset made publicly available by MNIST (Mixed National Institute of Standards and Technology) acquired from

Kaggle.com web site. The dataset contains 42,000 rows of classified data each row containing 785 (including the label) columns with each cell representing the intensity of the pixel (0-255) of the corresponding pixel. The images themselves are in gray-scale in order to facilitate computations and increase the accuracy.

Furthermore, the authors have decided to pick this specific dataset due to its relatively simplistic nature. Nonetheless, it still presents a problem attractive enough due to its many uses in the real world.

Figure 15 Sample pictures of the digits in the dataset

The total amount of columns can be calculated by multiplying the height x width in pixels. In the case of this particular dataset the images are 28 x 28 which is 784 (Not counting the label) as seen in Figure 15.

Finally, it has also been decided that the dataset would be divided with the following proportions for training, cross-validation and test respectively, 60/20/20. It is important to highlight that said proportion is, by no means, absolute. The proportion, however, was considered appropriate due to preliminary results achieved with them.

The reason for picking this particular set of features is due to their simplicity and the fact that non-experienced readers can easily understand it.

3.5 Description of implementation

3.5.1 Preamble

Since the dataset is large and artificial neural networks are usually computationally demanding it is worth mentioning the hardware and software that were used throughout the research. One of the computers is Asus i7 Haswell 2 GHz, 8 GB RAM, SSD drive, Windows 8.1 64x. The second computer used is Sony Vaio i3 1.7 GHz, 4 GB RAM, SSD drive, Windows 8.1 64x. Nowadays some specific libraries allows taking advantage of computations on top of the graphical processing unit, but that is not the case for us, so the basic setup and computations were on top of the CPU.

In the description of implementation, each version on a logical level without specific programming details due to the fact that source code is publicly available at github repository[2].

The structure and explanation of the folders can be found in Appendix 1. Due to the characteristics of the development project from software engineering point of view, there was no need to have extensive design documentation. This can be attributed to the fact that, the concept itself is more important than the code. Furthermore, the amount of code written is quite limited and does not need to be documented due to its simplistic nature. Such nature is made possible by built-in and third party modules available to all of the ecosystems discussed on this paper. However the conceptual design of the program is covered in Appendix 2 with a brief picture of what functions the program should contain.

3.5.2 Version 1

Data Processing

The first step is to load the data into the main memory. The file itself is a comma-separated-values (CSV) file.

```
> str(digits_dataset)
'data.frame': 42000 obs. of  785 variables:
 $ label   : int  1 0 1 4 0 0 7 3 5 3 ...
 $ pixel0  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel1  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel2  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel3  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel4  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel5  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel6  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel7  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel8  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel9  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel10 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel11 : int  0 0 0 0 0 0 0 0 0 0 ...
```

Figure 16 Example of the data representation, 42000 columns, 785 rows (784 pixel values + 1 label)

Figure 16 depicted above represents the data frame given to us by R. The header states that 42.000 observations exist, in other words, there are 42.000 columns. It also states that 785 variables exist which is basically a different way of saying that 785 features exist per sample. Each row represents the intensity of a given pixel for a particular label.

[2] https://github.com/Ytseboy/thesis-project-x

After loading the data into the main memory the data needs to be split, as mentioned earlier, into three sets: training set, cross validation set and test set. The dataset will be divided in a 60/20/20 ratio for each set respectively. The program itself performs the division of the dataset by using the following calculation:

<*c1 = m * 0.6; c2 = m*0.8*> where *m* is amount of rows in the full dataset, *c1* and *c2* are cut points.

After that point, taking into account that our target is a multiclass classification, where we have ten possible classes (digits from zero to nine), we need to represent the labels in a class indicator matrix. As the result, we get for all of the labels a binary matrix that represents the class of the particular label. Figure 17 illustrates an example of the class-indicator matrix. For each row in the table, all the cells have value zero except the one that corresponds to the class that particular row belongs. For instance on the first row, the value one is in the second position, meaning that for that particular the row belongs to the second class, which in our case is the digit (label) one.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	1	0	0	0	0	0	0

Figure 17 Class indicator matrix example

Note that in figure 17 each column represent a class, which is the range of values that a given row can take. There are 10 columns (0 – 9). Consider, for instance, the second row which has the value 1 for the class V1 and 0 for the remaining class. Based on that, one can say that row 2 represents the first class, which corresponds to digit 0.

Having that in mind, the output or, in other words, predictions of the future model, will also be in the form of a class indicator matrix where the values express a level of certainty (in decimal) that a given image belongs to a certain class. Evidently, the class indicator matrix itself still has to be transformed so that it gives the desired label output (the digit itself). For example, consider again another sample for which we do not know the value. We run our algorithm which gives us back 0.98342 for the column V2 and some smaller values for other columns. This result basically implies that the entry (image) with confidence of 98,34

% represents the second class which corresponds to digit one. The level of certainty the algorithm would have that our unlabeled sample belongs to that class.

Training the model

Previously, the basic training parameters, such as amount of hidden units, maximum optimization steps and, for some libraries, threshold of the partial derivative, were chosen manually.

Afterwards, the process of learning, in technical terms, can be described as the optimization of the cost function with respect to the network weights. The required input for the optimization function are the cost function, the initial network weights, the features and the class-indicator matrix as a classifier target.

Model performance

In order to evaluate the model performance for the basic level, we need to measure the training fit and the test accuracy, which is done by comparing the actual model output against the expected model output. The difference between the metrics is that the first evaluates how well the learning process went and how well the model fits the training set of the data. The second one then evaluates the ability of the model to do predictions, in other words, how well the model is able to generalize on the data. Both are calculated as the average error of the classifier, which can be obtained by dividing the amount of correct classifications by the amount of rows.

3.5.3 Version 2

As described, in the results section version 1 sub-chapter, a model that uses the full image resolution (28x28) means that we have 784 features. First of all that might be seen as “curse of dimensionality” described earlier in the theoretical background section “Problems that affect accuracy”. Additionally, on a logical level the model with 50 hidden units there would be 785*50 + 51*10 = 39760 network weights. Which means that the cost function exists in a 39761 dimensional space, which may potentially open room for a possible local minima which could constitute a “Failed optimization” problem.

Although, the problems mentioned above were perceived at the theoretical stage, it was not evident that they would have such an impact. For mitigation purposes, the initial implementation plan was adjusted, and phase IV, which was reserved for possible data permutations, was executed after the first phase.

Data permutations

The first approach to simplify the complexity of the data can be considered as a scientific method called “Principal Component Analysis” (PCA). PCA is a method of reducing dimensionality of the data, while losing the least amount of information that the data contains. The method, in simple words, can be formulated as finding a line that minimizes the sum of the distances between a point and that line for all the points in a given n-dimensional space. From the implementation perspective, a function ‘princomp(data)’ was used, that is part of a package ‘stats’ in R. After that, five components were used as new features for training the model and two first components were used for 2D data representation as seen in figure 18.

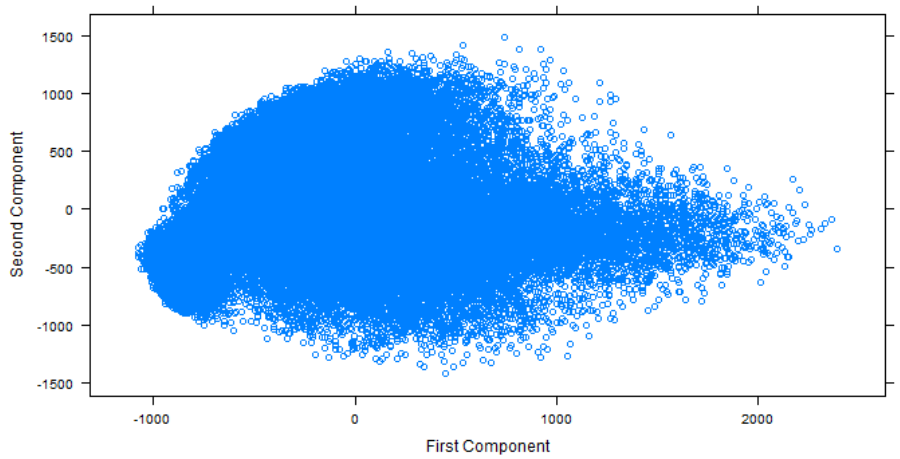

Figure 18 2D representation of the digits dataset based on PCA method

Concurrently, another more pragmatic approach was used to reduce the complexity of the data. For each sample in the dataset, the image was divided into 2x2 matrices and for each block, the pixel with maximum density was chosen as the most significant one. Following that, the significant pixel was then added to a new matrix. That procedure gave as an output the dataset with 14x14 resolution images, illustrated in figure 19, meaning that we only have 196 features for each row in the dataset.

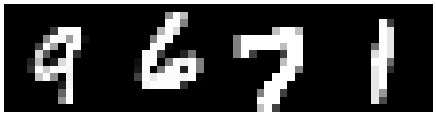

Figure 19 14x14 image samples

It is essential to highlight that, in general, even after the reduction, the images are still readable meaning that there was no practical loss of information.

3.5.4 Version 3

In this version, the implementation was mostly focused on establishing the model evaluation metrics, in pursuance of fully understanding the weak points in the model in addition to what changes should be considered for the upcoming versions.

Model evaluation

The first metric that was implemented was the cost function over the iterations, as described in the Theoretical background. The values of cost function for each of the optimization steps were collected from the optimization function and then plotted as a graph.

The second metric that was implemented was the learning curve over the amount of training data. For collecting the training fit and validation accuracy, a simple for-loop control flow was implemented where several models were trained on an increasing size of training data. For example, the first model was trained using 100 entries of the training data and for every subsequent model 100 extra entries were added, as in an arithmetic progression, until the counter reached the value which represents the number of rows of the whole training set (60 % of the dataset). Note that each iteration validation makes use of the full validation set (20 % of the dataset).

The third metric that was implemented was the confusion matrix where one might quickly see how well the classifier is capable of recognizing a certain class. The matrix itself is a plot of actual versus expected classifications

3.5.5 Version 4

The target for current iteration was to discover optimal model state.

Model calibration

As described in the theoretical background section “Algorithm debugging and model optimization” the first step to take with neural networks is to identify the optimal amount of hidden units for a hidden layer in the neural network. That was achieved by using a simple for-loop control flow that iterated over several models using a different amount of hidden units, starting at 50 and adding 50 more for every subsequent iteration of the loop all the way up to 1000. For each iteration both, the training fit and validation accuracy were recorded later comparisons.

Following that, the model with the optimal amount of hidden units was sent to a different for-loop control flow that applied different regularization values on it, starting from 0 up to

10 using an array of predetermined values (0.1, 0.3, 0.7, 1.9…). As in the previous step, the training fit and validation accuracy were recorded.

3.5.6 Version 5

For this version, only MATLAB was used, as explained in the Results, Version 4. To acquire the final results we yet need to stabilize the model. The last step considered was to approach the noise in the data and probabilistic momentum of random weights initialization in artificial neural networks.

Model averaging and stabilization

At first, the model was calibrated on a very detailed level meaning that several models were trained using different network parameters; however, their difference was very little. After that ratio of training set relatively to the full, dataset was increased up to 70 %, which left 15 % for each validation and test set.

The next step taken was about training several (10) similar models and averaging their output. This was achieved by looping the training process and basic evaluation into for-loop control flow.

3.6 Results

3.6.1 Version 1

At first, a rough implementation with 50 hidden units was done in Octave. At that point, 2000 rows (images) were used as training set and the remaining rows were used for validation. As a result, figure 20 presents a screenshot, where the program's performance relative to the amount of data used was 86 % for training fit and 84 % for test accuracy.

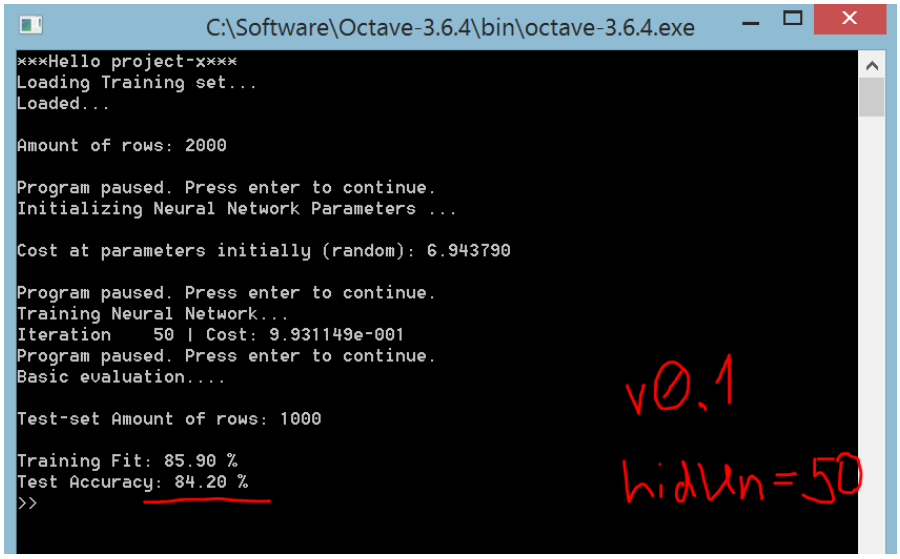

Figure 20 Screenshot of Octave prototype

Finally, a submission file was created with the objective of checking how Kaggle's own result compares to the result given by the algorithm. The evaluation, depicted in Figure 21, was 83%, which is close to the program's self-evaluation.

Figure 21 Screenshot of Kaggle Handwritten digits classification competition leaderboard

Afterwards, the first version of the program, using R and the package neuralnet, was implemented. The model was trained using 50 hidden units against 3000 rows of data and, following the pattern, the remaining rows were used for the model accuracy evaluation. As the result, program reported 89 % for training fit and 76 % for test accuracy.

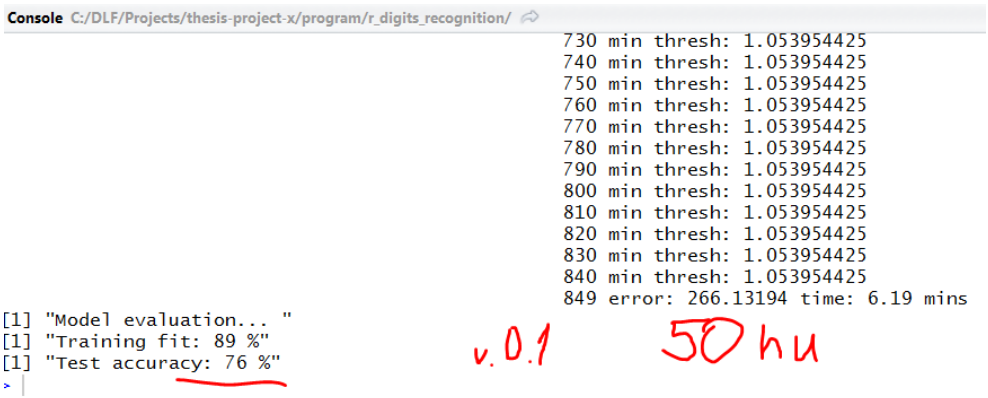

Figure 22 Screenshot of R_neuralnet first version

The preliminary results, emphasized in Figure 22, were deemed sufficient, as this is only the first iteration of the implementation. It is worth of a mention, that the program's execution times were an issue. Using only a small fraction of the data, the execution time was

over 6 minutes - which is not desirable considering that the whole dataset is over 40.000 rows.

At that point, alternative artificial neural networks packages were considered. The second package used, called ‘nnet’, also performed rather poorly. In fact, running it with the same dataset (3000 rows) an exception was thrown due to a memory overflow.

A third library was also taken into use, called ‘RSNNS’. RSNNS was able to train the model, yet, like neuralnet, the time of execution for the program was about 6 minutes.

Analyzing the preliminary results, it was underlined that the most significant problem with the tested packages is the relative complexity of the data (28x28 pixels = 784 columns), which lead to an increased complexity of the model, that with 50 hidden units consists of 785*50 + 51*10 = 39760 weights. For this reason, the Research group decided to switch the phases of the implementation, continuing with Data permutations, to reduce the complexity of the data.

3.6.2 Version 2

After applying Principal Component Analysis on the full dataset, the 5 first components were selected as derivative features, and the program with R_neuralnet package was executed. The model with 50 hidden units resulted in having 70 % of the training fit and 69 % of the test accuracy, as highlighted in figure 23. That would be considered as an acceptable intermediate result, however the time of execution for the program still were an issue - it took around 10 minutes to train the model. Taking into account the current relative simplicity of the model (6*50+51*10) = 810 weights, time of execution was not decreased much comparing to the results of the version 1.

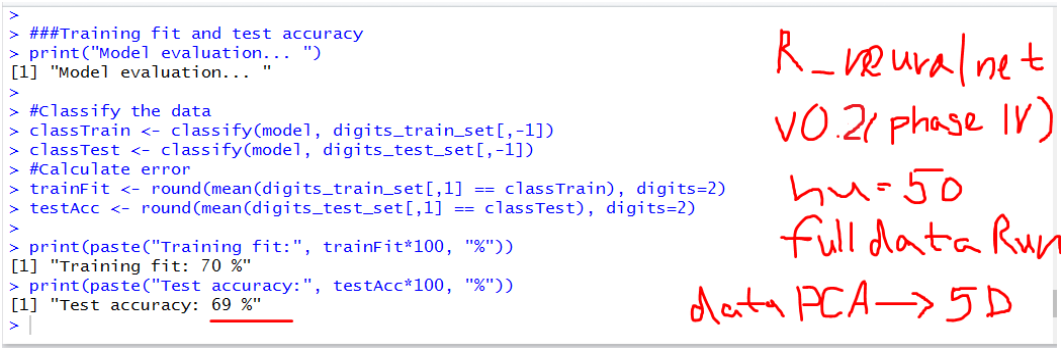

Figure 23 Screenshot of model evaluation after PCA method

Subsequently, after the previous performance, a different ecosystem (MATLAB/Octave) was tried. The first observation is that there was a big discrepancy in the time taken by both applications to load the full dataset. On top of that, there was also a gap in the hardware resources used by both ecosystems.

For this particular test, the dataset used contained the 14x14 images described in the implementation Version 2. The set was also divided according to the pre-established ratio (60/20/20) for training, validation and test respectively. Without any modifications to the remaining parameters in the accuracy increased to 94% as shown in figure 24.

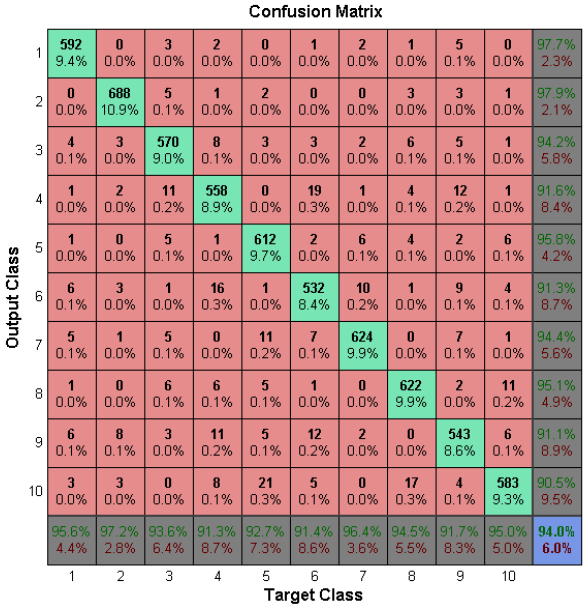

Figure 24 Test accuracy screenshot in plot of Confusion matrix

It is also important to emphasize, that the model finished training after 85 iterations and 34 seconds, which was a significant performance boost over R.

For this reason, the authors have decided to switch the focus of the attention from the R ecosystem to the more efficient MATLAB/Octave ecosystem.
In the second iteration with MATLAB/Octave, the amount of Hidden Layer Units was doubled from 50 to 100. The confusion matrix below depicts the results.

According to the confusion plot there was an increase of 2.3% compared to the previous iteration. As for the computation time, it remained roughly around the same

3.6.3 Version 3

After the implementation, the plots and metrics in version 3 were collected.

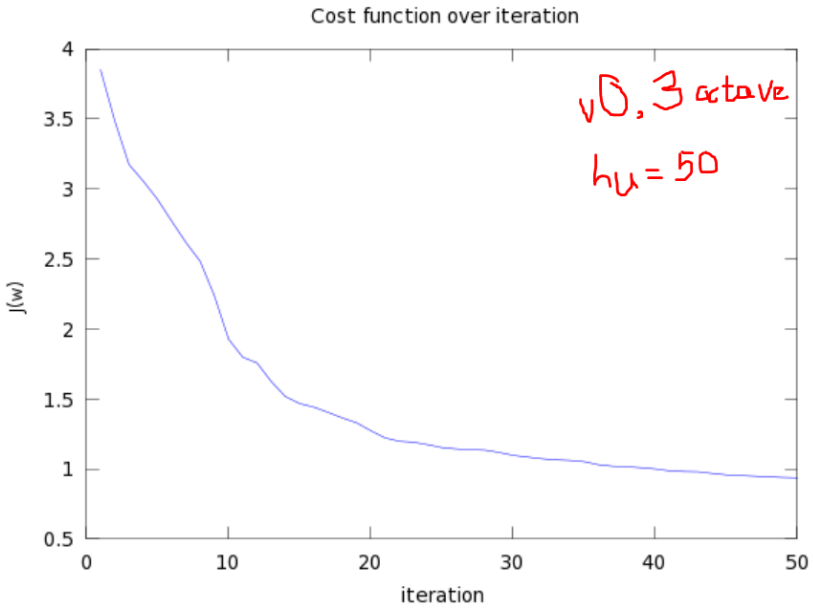

Figure 25 Cost function over optimization step plot

Figure 25 above represents good optimization results. First, the cost function value is decreasing on the whole interval meaning that the optimization algorithm with the given parameters is capable of optimizing the function. However, as it can be seen, the slope of the function is not yet close to zero, meaning that the function at its' end is not approximating a horizontal line - the conclusion can be made that that cost function is not yet at its near-optimal point, thus by increasing the amount of optimization steps - a better training fit can be achieved. The second metric learning curve was then executed.

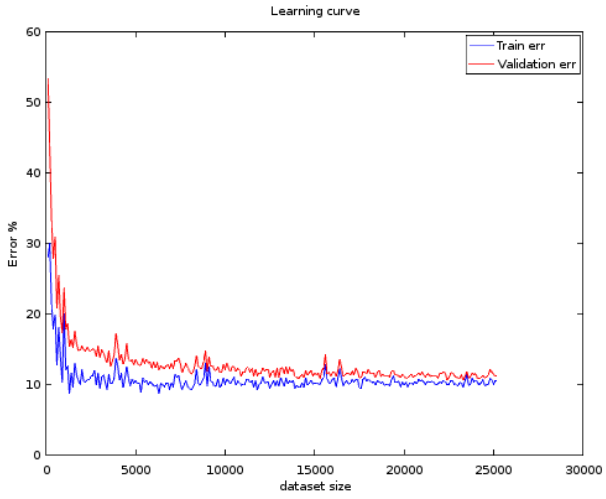

Figure 26 Learning curve as error rate over training set size

In figure 26, it might be clearly seen that with 50 hidden units the model right from the start running into error due to high bias, due to the relative distance between two. That means that the model fails to fit even the training data, thus the model complexity should be increased.

The third plot represents ability of classifier to recognize a certain class.

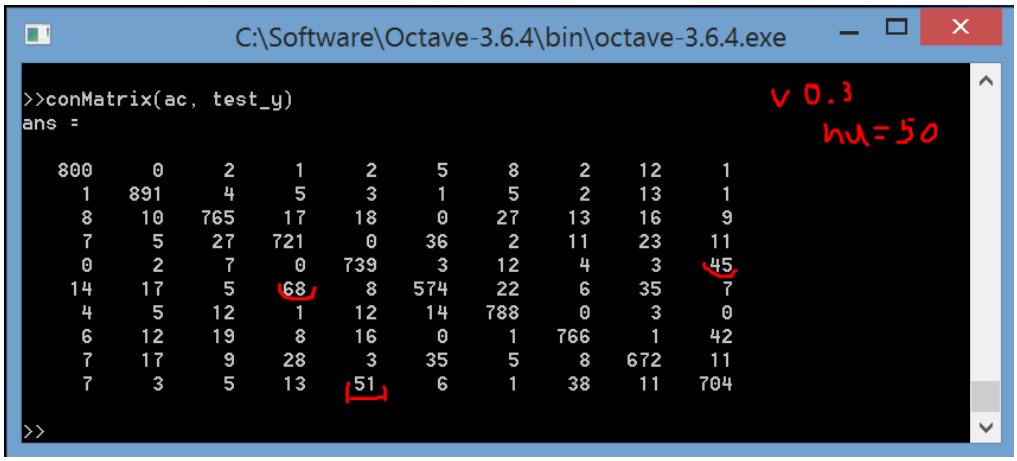

Figure 27 Confusion matrix

In figure 27, the numbers aligned in the diagonal represent matches between actual versus expected classification, everything that is out of diagonal is a deviation. So far as preliminary results using 50 hidden units the amount of deviations is not surprising. However, one particular observation might be made is that Classifier recognized 68 digits-five as three, 51 digits-9 as 4 and 45 digits-4 as 9. The last two are especially interesting telling us that, now the classifier has difficulties to distinguish digits 9 and 4.

3.6.4 Version 4

Even though the process of training several models with a different amount of hidden units was time consuming, the time spent was still productive. Altogether, the process lasted about 3 hours and in the end, the program provided a plot of training and validation error rates (percentage) over the amount of hidden units used.

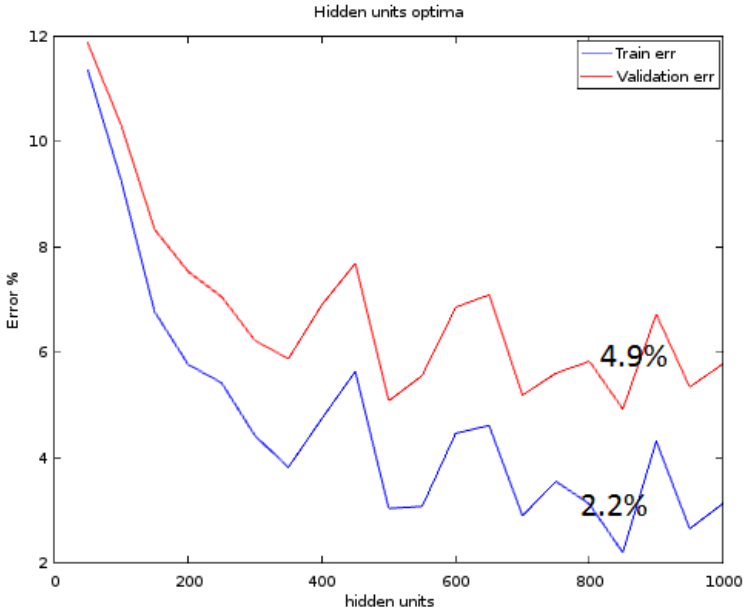

Figure 28 Hidden units model calibration

According to figure 28, the model with 850 hidden units showed to be the optimal for Octave implementation providing minimum validation error. Right after that regularization, the parameter calibration was executed and again it provided a graph of error rate over regularization value as clarified by figure 29.

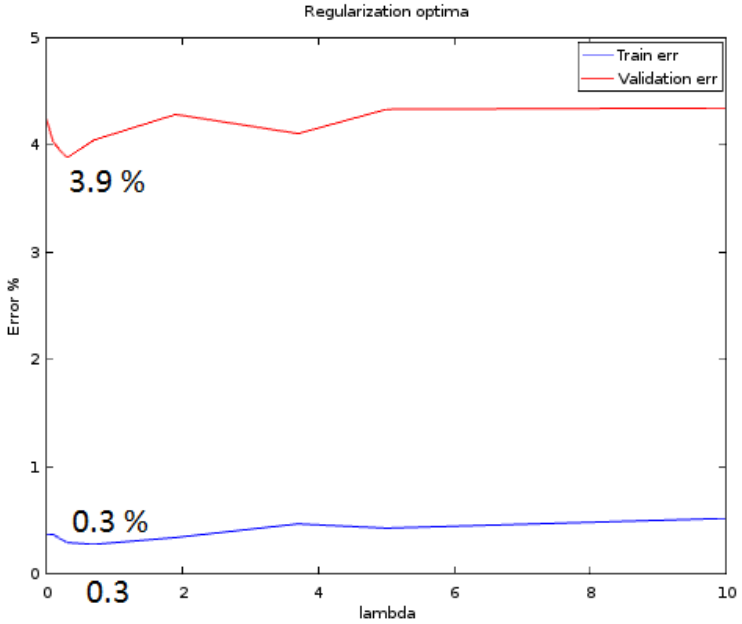

Figure 29 Regularization parameter model calibration

Although the difference is not very significant, the graph shows that 0.3 provided a slightly lower validation error. Putting it together, the calibrated model was trained and the evaluated result was 95,6 % of accuracy, which figure 30 illustrates.

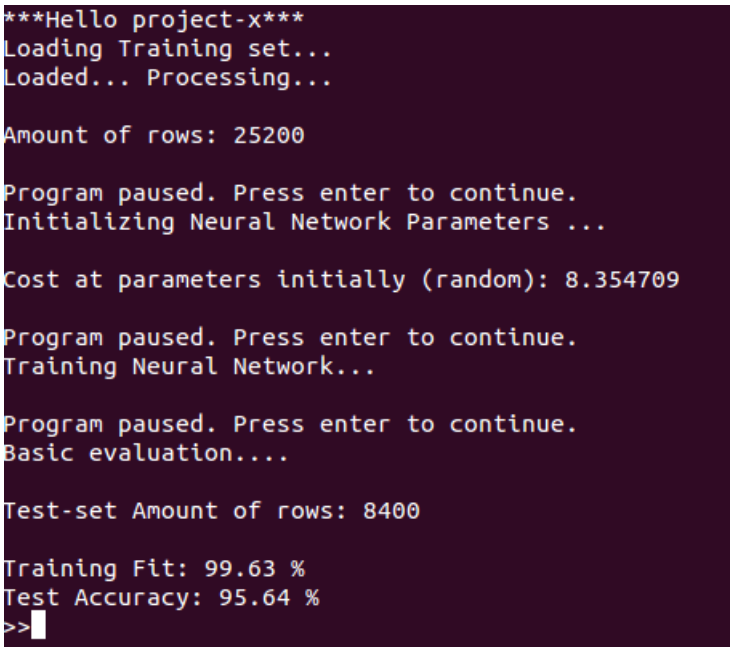

Figure 30 The program screenshot after training calibrated model

Two more attempts of more precise calibration runs were endeavored, using a smaller range and difference between the amount of hidden units, range from 810 up to 890 with increment of 10 and then range from 841 up to 859 with increment of 1. Although the resulting optimum was encountered at 845 hidden units, Octave's implementation reached the plateau of its accuracy at 96%. Based on the results it was decided to proceed using only one tool - MATLAB since already at earlier stages it showed a better performance (Research results version 2) and requires a less complex model.

3.6.5 Version 5

By the final calibration in MATLAB, the following network parameters were chosen: 194 hidden units and 0.575 regularization value. That set up showed itself to be the optimal and resulted in 99,8 % and 97,5 % for the training fit and test accuracy respectively.

For the final countdown, we went all-in and made sequentially averaged models with 10, 20, 30 and 50 neural networks. Training each network was taking about 6 minutes, so for 50 networks execution time was about 5 hours. The averaged model with 50 networks evaluated training fit of 99,8 % and test accuracy of 99,7 % as revealed by figure 31.

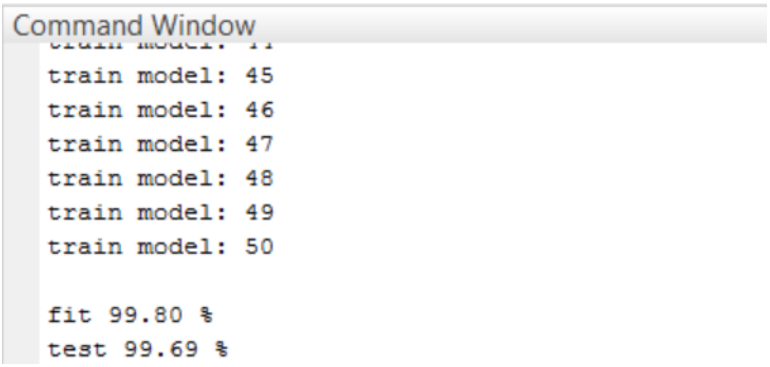

Figure 31 Screenshot of program, which uses 50 ANNs for providing, averaged output

Right after that, the Kaggle submission was done for the sake of validating the evaluation. In figure 32 we could observe accuracy of 97,6 %

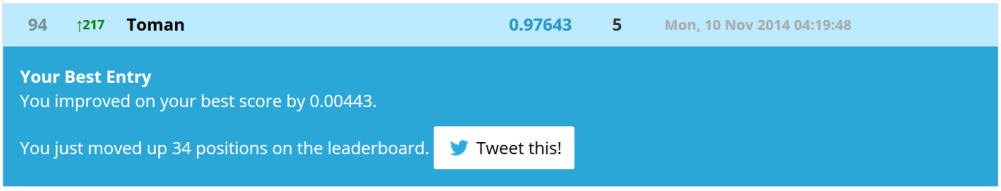

Figure 32 Final Screenshot of Kaggle digits recognition competition leaderboard

Even-though Kaggle web site competition was out of scope of this research, it still was used for the sake of result trustworthiness check and sort of baseline. Before with submissions, the general trend was that Kaggle leaderboard score is slightly (less than one percent) lower, than program self-evaluation. However, in the final submission we met a deviation of about 2 %.

3.7 Summary

Primarily, the first thing one should note is that data exploration in addition to pre-processing are not only important but also vital for the outcome of the computations. It has been found that raw data can hinder the speed with which the computations are performed. In addition, data processing can also facilitate further interpretation and reduction of the collected samples.

On top of that, the selected ecosystem really influences the time and accuracy of the outcome. From the tools utilized throughout the paper, it has been observed that R had the worst performance among the selected tools. These two factors onto themselves should already be enough to obtain a satisfactory result level. However, as a final step one could also theoretically increase the accuracy of the output by systematically tweaking parameters in the neural network.

Finally, other methods can be still utilized to further boost the accuracy. In this particular case, training several networks is much more time consuming and boost of accuracy is not very significant, still performance of averaged model was always better than usage of a single network. That being said, other methods could also have been used in its stead, but these are outside the scope of this thesis. Figure 33 summarizes the results over the iterations obtained by each tool.

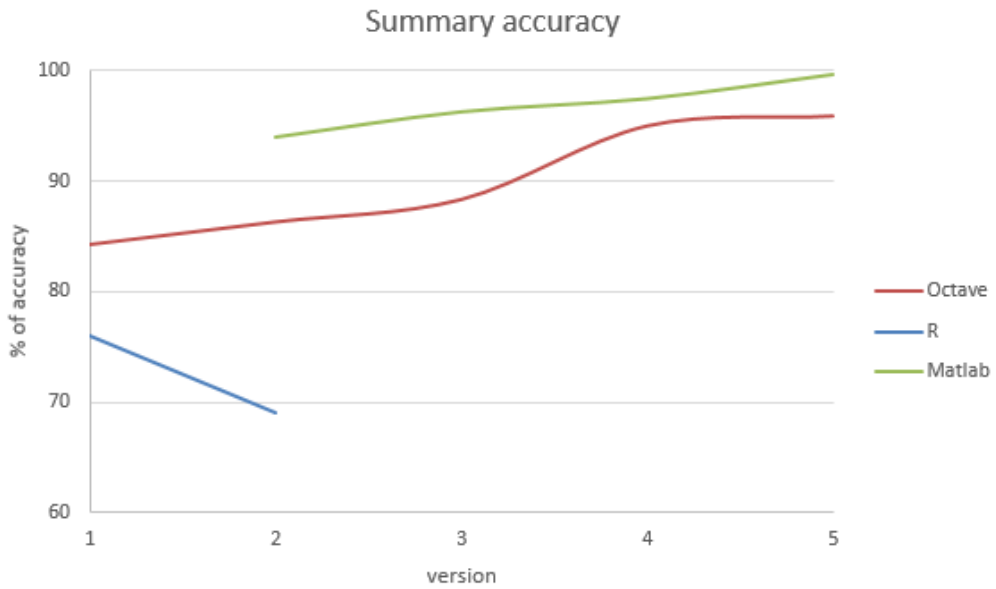

Figure 33 Accuracy performance of different tools over iteration

This section, finalizes the second part of this paper, the empirical part. Collection and evolution of the results presented up to this point will be used as a basis for the final part of the thesis.

4 Discussion

4.1 Considerations of results

The obtained results display the efficiency of Machine Learning, in particular, neural networks, in the resolution of problems that cannot be solved by common programming standards. It is in complex situations like this that Machine Learning thrives. The accuracy obtained with the algorithm was overall satisfactory and it highlights the potential for this kind of technology in the academic and practical environments.

It is also important to consider that only relatively basic techniques were used to achieve such results. With more processing power, advanced neural networks architectures and techniques that are more complex one could theoretically elevate the accuracy to 100%.

4.2 Trustworthiness of the research

The content pertaining to the theoretical background of the research was carefully written and can be backed by the scientific method along with extensive mathematical reasoning.

Furthermore, Kaggle submissions verify the accuracy of the results found in the empirical part by giving the accuracy of one's personal results against the actual results stored on the server. The results given by Kaggle, serve to reinforce the accuracy of the results obtained.

Last but not least, the results achieved throughout this research are all reproducible. Obviously, that would only apply as long as the described instructions are carefully followed.

4.3 Suggestions for further work

For further work, one may explore the data pre-processing and analysis processes. From the results, it became self-evident that both these areas play a vital role in the overall outcome of the algorithm. Initially, one may be lead to believe that these processes are trivial, but as the number of features and samples increase so does the importance of those areas. For example, one could consider noise reduction techniques or dimensionality reduction such as PCA, eigenvectors and so forth.

4.4 Conclusion

The previous sections allow us to conclude with a high rate of certainty that Artificial Neural Networks can indeed be used to recognize hand-written digits with a significant level of

confidence. Furthermore, additional methods can be used in order to increase the accuracy and overall efficiency of the neural network.

As for the methods, the Results section clearly indicates that there is no quintessential method, which boosts the overall accuracy. Instead, an array of methods are collectively used, tested and their total summation finally offers an observable increase in the final accuracy.

Moreover, one of the most important points one should take after reading this paper is the special attention one should pay to the dataset and data processing methods. Before even implementing the algorithm the data ought to already be processed. This cannot simply be emphasized enough. Referencing back to the results section, not only data processing improves computation efficiency, but also it reduces the error rates. As for the data processing itself, it is very important to known the context of the data before performing any sort of processing methods. In addition, adjusting Artificial Neural Networks properties (e.g. regularization value, amount of hidden units) also notably increased the accuracy.

Finally, it has been found that the algorithm itself although important is only a piece of the overall puzzle. When dealing with Machine Learning, one should consider the factors described above and also consider the ecosystem and tools used. The final benchmarks show that some tools perform much more efficiently with the same processing power under the same computer configuration. The impact was significant enough to justify a complete change of ecosystems, not once, but twice.

Briefly, the objective set at the beginning of the research was met. However, relatively speaking, the accuracy achieved could have been higher. Nonetheless, all the questions inquired at the Introduction section have been successfully answered.

4.5 Evaluation of the thesis process and one's own learning

One thing we definitely learnt is that one should know its enemy. Meaning that there are many factors that cannot be accounted beforehand, unless one have extensive experience. We cannot know ahead of time that the execution time and lack of processing power would have such a considerable impact. In addition, we were deceived by the community's opinion about the R programming language for data mining and machine learning.

On top of that, In general in supervised learning we are dealing with a function approximation of an unknown target function, which creates a lot uncertainty about what is happening under the hood. Therefore, early prototyping and extensive evaluation should be always in place.

Regarding the thesis process itself, there were no major obstacles along the way. Diligence, focus and organization played an important role in the thesis. Overall, it should be emphasized that one should learn how to plan properly before tackling a project of such a large scope like a thesis.

References

Bellman, R 1961 Adaptive Control Processes: A Guided Tour Princeton University Press

Domingos, P. 2012. A few useful things to know about machine learning. Department of Computer Science and Engineering University of Washington.

Fortmann-Roe, S. 2012. Understanding the Bias-Variance Tradeoff. URL: http://scott.fortmann-roe.com/docs/BiasVariance.html Accessed: 06 October 2014.

Hamilton, L. 2014. Six Novel Machine Learning Applications. Forbes. URL: http://www.forbes.com/sites/85broads/2014/01/06/six-novel-machine-learning-applications/ Accessed: 25 September 2014.

Heaton, J. 2008. The Number of Hidden Layers. URL: http://www.heatonresearch.com/node/707 Accessed: 10 October 2014.

Manning, C., Raghavan, P. & Schütze, H. 2008. Linear versus nonlinear classifiers. URL: http://nlp.stanford.edu/IR-book/html/htmledition/linear-versus-nonlinear-classifiers-1.html Accessed: 19 October 2014.

Mitchell, T. 1997. Machine Learning. McGraw Science/Engineering/Math.

Ng, A. Advice for applying Machine Learning. Stanford University. URL: http://see.stanford.edu/materials/aimlcs229/ML-advice.pdf Accessed: 09 October 2014.

Scikit-learn (Machine learning library for python, official web site). URL: http://scikit-learn.org/stable/ Accessed: 01 October 2014.

Spruyt, V. 2014. The Curse of Dimensionality in classification. URL: http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/ Accessed: 07 October 2014.

Srihari, S. Regularization in Neural Networks. URL: http://www.cedar.buffalo.edu/~srihari/CSE574/Chap5/Chap5.5-Regularization.pdf Accessed: 11 October 2014

Appendices

Appendix 1. Github Repository: URL & folder structure

URL: https://github.com/Ytseboy/thesis-project-x/

Repository Structure:.

```
│   .gitignore
│   README.md
├───design
│       nnet_plot1.jpg
│       program-pipe.txt
│       wt_ex.jpg
│
├───misc
│   │   classIndicatorMatrixExample.png
│   │   OctaveKaggle_v0.1_20141012.png
│   │   Octave_v0.1_20141005.png
│   │   Octave_v0.1_20141025.png
│   │   Octave_v0.3_20141030_conMatrix.png
│   │   Octave_v0.3_20141030_CostFuncPlot.png
│   │   Octave_v0.3_20141030_learningCurve2000.png
│   │   Octave_v0.3_20141030_learningCurve25200_hu50.png
│   │   Octave_v0.4_20141104.png
│   │   Octave_v0.4_20141104_calHUplot.png
│   │   Octave_v0.4_20141104_calRegplot.png
│   │   Rneuralnet_v0.1_20141025.png
│   │   Rneuralnet_v0.1_20141025_PCA_5diamensions.png
│   │
│   ├───images_14x14
│   │       label1.png
│   │       label10.png
│   │       label11.png
│   │       label12.png
│   │       label13.png
```

```
|   |
|   └─────images_original_28x28
|           dataExample0.png
|           dataExample1.png
|           dataExample3.png
|           dataExample7.png
|           dataExample8.png
|           dataExample9.png
|
├─────original-data
|       README.md
|       test.csv
|       train.csv
|
└─────program
    |   reducedTestForKaggle(updated).csv
    |   reducedTestForKaggle.csv
    |   reduced_14x14_fullNoHeaders.csv
    |   reduced_5_rows_42000.csv
    |   testForKaggle.csv
    |   trainLoadTry.csv
    |   trainLoadTry_headers.csv
    |   train_1to5000.csv
    |   train_full.csv
    |   train_headers_1to5000.csv
    |
    ├─────Matlab
    |       multiple_nn.m
    |
    ├─────Octave
    |       assert.m
    |       calibrationHU.m
    |       calibrationReg.m
```

```
│       conMatrix.m
│       displayData.m
│       fmincg.m
│       kaggleAnswer.csv
│       learningCurve.m
│       main.m
│       makeAnswer.m
│       nnCostFunction.m
│       octave-core
│       plotCostFunction.m
│       predict.m
│       randInitializeWeights.m
│       sigmoid.m
│       sigmoidGradient.m
│       trainModel.m
│
├─────r_neuralnet
│       .RData
│       classify.R
│       main.R
│       pca.R
│       r_digits_recognition.Rproj
│
├─────r_nnet
│       classify.R
│       main.R
│       try1.Rproj
│
└─────R_rsnns
    .RData
    documentation_example.R
    main.R
    R_rsnns.Rproj
```

Appendix 2. Program pipe (conceptual design)

number means the iteration

1.1 load and divide dataset [train, cross-validation, test]
1.2 set up network parameters
1.3 Learn [optimization], find weights that gives minimal cost function value
1.4 measure training fit & test accuracy

2.1 Image resolution reduction (28x28 --> 14x14)
2.2 PCA (dimensionality reduction)

3.1 Learning curve plot
3.2 Cost function
3.3 Confusion matrix

4.1 Model calibration Hidden units
4.2 Model calibration Regularization

5.1 Model Averaging and Stabilization

0.1 Display data (image)
0.2 Make Kaggle submission

Glossary

Artificial Neural Networks (ANN): (*Multi-layer perception* in some literature) Computational model based on biological neural networks concept. It is commonly used to estimate functions that depend on a large amount of inputs and are not usually known.

Classification: [statistics] Process of identifying to which class/category a given observation belongs.

Computer vision: is a multidisciplinary field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information

Hidden Units: Number of nodes in the hidden layer of a given Artificial Neural Network

Kaggle: Website (kaggle.com) for data prediction competitions.

Machine Learning: Multidisciplinary field which studies and deals with algorithms which are capable of learning from data.

MATLAB: MATLAB stands for **MAT**rix **LAB**oratory. It is a numerical computing environment and a programming language.

MNIST dataset: Dataset of handwritten digits, comprising 60.000 training examples and 10.000 test examples.

Octave: High level interpreted language used for numerical computations.

PCA: Principal Component Analysis. One of the methods to decrease dimensionality of data.

R: Programming language and environment used for statistical computing and graphics.

Regularization: The process of introducing additional information to avoid overfitting.

Supervised Learning: The process of inferring a function based on labeled data.